Oestrich Pascal

LB 450

16.01.2024

Inhalt

[Teilaufgabe 1 3](#_Toc156291687)

[*Projektbeschreibung 1.1* 3](#_Toc156291688)

[*Anforderungen 1.2* 3](#_Toc156291689)

[*Testfälle 1.3* 3](#_Toc156291690)

[Teilaufgabe 2 7](#_Toc156291691)

[*Beschreibung 2.1* 7](#_Toc156291692)

[*Nachweis 2.2* 8](#_Toc156291693)

[Teilaufgabe 3 13](#_Toc156291694)

[*Erklärung IoC 3.1* 13](#_Toc156291695)

[*Testarten 3.2* 13](#_Toc156291696)

[*Dependency in meinem Programm 3.3* 15](#_Toc156291697)

[Teilaufgabe 4 17](#_Toc156291698)

[*Code Review 4.1* 17](#_Toc156291699)

[*Verbesserung des Codes 4.2* 19](#_Toc156291700)

[Teilaufgabe 5 23](#_Toc156291701)

[*Unittest 5.1* 23](#_Toc156291702)

[Teilaufgabe 6 25](#_Toc156291703)

[*Erklärung Selbstgemachter logischer Fehler 6.1* 25](#_Toc156291704)

[*Unterschied zwischen Autos und Lokal 6.2* 27](#_Toc156291705)

[*Watches 6.3* 28](#_Toc156291706)

[*Debuggen 6.4* 28](#_Toc156291707)

[Teilaufgabe 7 33](#_Toc156291708)

[*Klasse: Square 7.1* 33](#_Toc156291709)

[*Klasse: Circle 7.2* 39](#_Toc156291710)

[Teilaufgabe 8 45](#_Toc156291711)

[*Implementierung Fehlerumgang 8.1* 45](#_Toc156291712)

[*Testprotokoll: 8.2* 45](#_Toc156291713)

[*Testfazit 8.3* 46](#_Toc156291714)

## Teilaufgabe 1

### *Projektbeschreibung 1.1*

In diesem C# in OOP schreibe ich ein Programm, das die Fläche, Umfang etc. von einem Rechteck oder einem Kreis mit vor eingegebener Seitengrösse berechnet und ausgibt.

Als Nutzer kann man auswählen, ob man ein Rechteck oder einen Kreis als Form nutzen möchte. Dazu muss man bei den jeweiligen Formen (Rechteck) Seitengrösse und bei (Kreis) nur den Umfang selbst eingeben.

Das Programm hat ein Interface, 5 Klassen und 4 Mock-klassen inklusiv «Program.cs» und ist im OOP geschrieben.

### *Anforderungen 1.2*

|  |  |  |  |
| --- | --- | --- | --- |
| **US-№** | **Verbindl-ichkeit** | **Typ** | **Beschreibung** |
| 1 | Muss | Funktion | Als ein Benutzer möchte ich die Länge und Breite eines Rechtecks eingeben können,  um dessen Fläche und Umfang zu berechnen. |
| 2 | Muss | Funktion | Als ein Benutzer möchte ich den Radius eines Kreises eingeben können,  um dessen Fläche und Umfang berechnen zulassen. |
| 3 | Muss | Funktion | Als ein Benutzer möchte ich, dass die Berechnungen korrekt sind. |
| 4 | Muss | Qualität | Als ein Benutzer möchte ich klare Anweisungen für die Eingabe von Massen erhalten,  um Fehler zu vermeiden. |
| 5 | Muss | Funktion | Als ein Benutzer möchte ich das,  dass Programm mit Fehlereingaben umgehen kann. |
| 6 | Muss | Funktion | Als ein Benutzer möchte ich, dass meine Eingaben auf Gültigkeit überprüft werden,  um Berechnungsfehler zu vermeiden. |
| 7 | kann | Qualität | Als ein Benutzer möchte ich eine klare und verständliche Ergebnisausgabe,  um das Programm übersichtlicher zu gestalten. |
| 8 | Muss | Rand | Das Programm ist in VisualStudio und mit OOP geschrieben. |

### *Testfälle 1.3*

***Testfall 1.1: Flächen- und Umfangsberechnung für Rechteck***

* ***Testfallnummer:*** *1.1*
* ***Anforderungsnummer:*** *1*
* ***Voraussetzungen:*** *Das Programm wurde gestartet, Rechteck wurde gewählt.*
* ***Eingabe:***

*2: 10*

*3: 5*

* ***Ausgabe:***

*1: "Geben Sie die Daten von den Attributen ein. "*

*4: "Oberfläche: 50"*

*5: "Umfang: 30"*

***Testfall 2.1: Flächen- und Umfang Berechnung für Kreis***

* ***Testfallnummer:*** *2.1*
* ***Anforderungsnummer:*** *2*
* ***Voraussetzungen:*** *Das Programm wurde gestartet, Kreis wurde gewählt.*
* ***Eingabe:***

*2: 20*

* ***Ausgabe:***

*1: " Geben Sie die Daten von den Attributen ein. "*

*3: "Durchmesser: 6.366"*

*4: "Radius: 3.183"*

*5: "Oberfläche 31.830"*

***Testfall 3.1: Validierung der Berechnungen***

* ***Testfallnummer:*** *3.1*
* ***Anforderungsnummer:*** *3*
* ***Voraussetzungen:*** *Das Programm wurde gestartet und Testfälle 1.1 und 2.1 wurden erfolgreich ausgeführt.*
* ***Eingabe:***

*1: (Keine weiteren Eingaben (Berechnungen wird von den vorherigen Testfällen bezogen))).*

* ***Ausgabe:***

*2: (Bestätigung, dass die Ergebnisse der Testfälle 1.1 und 2.1 korrekt sind.)*

***Testfall 4.1: Überprüfung der Benutzeranweisungen***

* ***Testfallnummer:*** *4.1*
* ***Anforderungsnummer:*** *4*
* ***Voraussetzungen:*** *Das Programm wurde gestartet, irgendeine Form wurde gewählt.*
* ***Eingabe:***

*1: (Benutzer folgt den Anweisungen des Programms, um Daten einzugeben.)*

* ***Ausgabe:***

*2: (Anweisungen sind klar und führen den Benutzer korrekt durch die Eingabe.)*

***Testfall 5.1: Fehlerbehandlung bei falscher Eingabe bei Rechteck***

* ***Testfallnummer:*** *5.1*
* ***Anforderungsnummer:*** *5*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: "abc"*

* ***Ausgabe:***

*2: "Ungültige Eingabe. "*

***Testfall 5.2: Fehlerbehandlung bei falscher Eingabe bei Rechteck***

* ***Testfallnummer:*** *5.2*
* ***Anforderungsnummer:*** *5*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: -10*

* ***Ausgabe:***

*2: "Ungültige Eingabe. "*

***Testfall 5.3: Fehlerbehandlung bei falscher Eingabe bei Kreisen***

* ***Testfallnummer:*** *5.3*
* ***Anforderungsnummer:*** *5*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: "abc"*

* ***Ausgabe:***

*2: "Ungültige Eingabe. "*

***Testfall 5.4: Fehlerbehandlung bei falscher Eingabe bei Kreisen***

* ***Testfallnummer:*** *5.4*
* ***Anforderungsnummer:*** *5*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: -10*

* ***Ausgabe:***

*2: "Ungültige Eingabe. "*

***Testfall 6.1: Überprüfung der Eingabevalidierung bei Rechtecken***

* ***Testfallnummer:*** *6.1*
* ***Anforderungsnummer:*** *6*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: 0*

*2: 10’001*

* ***Ausgabe:***

*3: "Ungültige Eingabe. "*

*4: " Ungültige Eingabe. "*

***Testfall 6.2: Überprüfung der Eingabevalidierung bei Kreisen***

* ***Testfallnummer:*** *6.2*
* ***Anforderungsnummer:*** *6*
* ***Voraussetzungen:*** *Das Programm wurde gestartet.*
* ***Eingabe:***

*1: 0.0000001*

* ***Ausgabe:***

*2: "Ungültige Eingabe. "*

***Testfall 7.1: Klarheit der Ergebnisausgabe***

* ***Testfallnummer:*** *7.1*
* ***Anforderungsnummer:*** *7*
* ***Voraussetzungen:*** *Testfall 1.1 und 2.1 wurden erfolgreich durchgeführt.*
* ***Eingabe:***

*1: (Übernahme der Ergebnisse aus den vorherigen Testfällen.)*

* ***Ausgabe:***

*2: (Überprüfung, ob die Ergebnisausgabe klar und verständlich ist.)*

***Testfall 8.1: Überprüfung der OOP-Prinzipien und Struktur***

* ***Testfallnummer:*** *8.1*
* ***Anforderungsnummer:*** *8*
* ***Voraussetzungen:*** *Quellcode ist verfügbar.*
* ***Eingabe:***

*1: (Durchführung eines Code-Reviews.)*

* ***Ausgabe:***

*2: (Bestätigung, dass der Code den OOP-Prinzipien entspricht und gut strukturiert ist.)*

## Teilaufgabe 2

### *Beschreibung 2.1*

**UML ohne Fehlerbehebung:**

Ein Bild, das Text, Diagramm, parallel, Reihe enthält.

Automatisch generierte Beschreibung

Dies ist das UML für mein Programm. Man sieht 9 verschiedene Klassen.

**Start:** Input für den User, welche Form (Kreis oder Quadrat er berechnen möchte)

**Controller:** ist die Oberfläche, von dort wird das Programm gesteuert.

**IShape:** Ist das Interface für die Klassen Square und Circle, wie auch für die Mocks.

**Circle:** Klasse für die Form Kreis.

**Square:** Klasse für die Form Kreis.

**MockStubCircle:** Ist die Mock-Klasse mit dem Typ Stub, die einen gewissen Inhalt der Klasse Circle testet.

**MockStubSquare:** Ist die Mock-Klasse mit dem Typ Stub, die einen gewissen Inhalt der Klasse Square testet.

**MockSpyCircle:** Ist die Mock-Klasse mit dem Typ Spy, die das Aufrufen von Methoden in der Klasse Circle dokumentiert (Ausgibt wie oft, welche Methode aufgerufen wurde).

**MockSpySquare:** Ist die Mock-Klasse mit dem Typ Spy, die das Aufrufen von Methoden in der Klasse Square dokumentiert (Ausgibt wie oft, welche Methode aufgerufen wurde).

Stub = gibt einen fixen vordefinierten ausgaben zurück.

Spy = protokolliert alle Methodenaufrufe.

Ich verwendete beide Typen für beide Klassen, um das Programm gleichermassen testen zu können. Das heisst, Circle hat ein Spy und ein Stub-Mock, das Gleiche wie bei der Klasse Square.

MockStubSquare und MockStubCircle: Diese Stub-Mock dienten dazu, das Interface IShape zu testen /simulieren, ohne eine echte Funktionalität für die Methoden CollectData(), ShowAttributes() und Calculate() zu implementieren. Es wird genutzt, um sicherzustellen, dass die Controller-Klasse korrekt mit einem IShape-Objekt umgeht, ohne sich auf das tatsächliche Verhalten anderer Klassen verlassen zu müssen und um die Code-Logik zu überprüfen.

MockSpyCircle und MockSpySquare: Das Spy-Mock wird verwendet, um zu protokollieren, welche Methoden aufgerufen wurden, ebenfalls um den Code besser nachzuvollziehen. Spy-Mocks protokolliert, welche und wie oft eine Methode aufgerufen wurde. Im Programm protokolliert der Spy-Mock, wie die MockSpyCircle und MockSpySquare aufgeruft werden und protokolliert ebenfalls alle aufgerufenen und nicht aufgerufenen Methoden auf.

### *Nachweis 2.2*

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungMan Findet die Mockings im Projektmappen-Explorer unter «TestProject»

**MockStubSquare**

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

**MockStubCricle**

**Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung**

**MockSpySquare**

**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung**

**MockSpyCircle**

**Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

Automatisch generierte Beschreibung**

**ControllerTest.cs**

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

## Teilaufgabe 3

### *Erklärung IoC 3.1*

Inversion of Control (IoC) = Designprinzip einer Software, es dient, um die Kontrolle über eine Klasse einer anderen Klasse zu geben. Dies optimiert die Zusammenarbeit mit Klassen im ganzen Programm. Statt dass jede Klasse seine Abläufe selbst steuert, wird die Kontrolle einer externen Klasse übergeben. Dies ermöglicht ein flexibleres und anpassungsfreundliches Programm.

Normalerweise kontrolliert jede Klasse ihre eigenen Abläufe, aber IoC wirft alles um. Es sorgt, dass die Kontroller von einer anderen Klasse an jene Klasse übertragen werden.

Dies wird typischerweise mit Dependency Injection umgesetzt.

Es gibt zwei typische Arten von Dependency Injection

* **Constructer Injection:** Dort wird die Kontrolle über den Konstruktor übergeben.
* **Setter Injection:** Dort wird die Kontrolle über einen Setter übergeben.

Je nach Programmlogik und Aufbau muss man eines von den zwei Dependency Injection auswählen.

Ich habe in meinem Programm den Constructor Injection verwendet. Der ist in der Klasse Controller zu finden. In der Klasse Start, in der Methode GenerateNeededObject() wird die vom User ausgewählte Formklasse (Square, Circle) dem Konstruktor Controller übergeben. Die Klasse Controller wird an sich wird im Program.cs initialisiert.

### *Testarten 3.2*

3 verschiedene Testarten:

Unittest, Integrationstests und Mocking.

**1. Unit Tests**

**Zweck:** Unit Tests sind da, um die Funktionalität einzelner Methoden oder Klassen getrennt von anderen Teilen des Codes zu überprüfen.

**Testmittel:**

* **Testframeworks wie NUnit (Testprojekt):** Diese Frameworks bieten eine Struktur für das Schreiben und Ausführen von Unit Tests, um das Verhalten zu überprüfen.
* **IDEs wie Visual Studio:** Für die Entwicklung und das Debugging der Tests.

**2. Integrationstests**

**Zweck:** Integrationstests prüfen, wie verschiedene Teile des Codes (z.B. Klassen und Methoden) zusammenarbeiten. In meinem Programm könnte man Testen, wie sich die Klassen miteinander verhalten.

**Testmittel:**

* **Testframeworks NUnit (Testprojekt):** Hier wird auch Dasselbe verwendet, um Integrationstests durchzuführen.
* **Mocking-Frameworks:** Zum Simulieren von Abhängigkeiten, die nicht Teil des zu testenden Abschnittes sind (z.B. Benutzereingaben).

**3. Tests mit Mocking (Fokus auf Benutzerinteraktion)**

**Zweck:** Tests, die Mocking verwenden, sind da, um das Verhalten von Teilen vom Code zu testen, die externe Abhängigkeiten haben, wie in meinem Fall Benutzereingaben.

**Testmittel:**

* **Mocking-Frameworks (Moq):** Ersetzen von Konsoleneingaben durch den Nutzer mit vordefiniertem Wert, um verschiedene Aktionen zu testen.
* **Testframeworks und IDEs:** Für das Schreiben und Ausführen der Tests.

### *Dependency in meinem Programm 3.3*

**Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot enthält.

Automatisch generierte BeschreibungClass Start:** Dort kann der User zwischen der Square und Circle-Form (Klasse wählen).

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

**Class Controller:** Und das ist die Constructor Injection

## Teilaufgabe 4

### *Code Review 4.1*

1. **Millersche Zahl**: Dieses Prinzip sagt, dass die Komplexität von Code und UML-Diagrammen klein bleiben sollte, 7±2 Dinge sein.
   * **Beobachtung**: Das UML-Diagramm sieht komplex aus, ist aber überschaubar. Im Code haben alle Klassen 7 oder weniger Attribute und 7 oder weniger Attribute.
   * **Verbesserung**: Im UML sollen die Mocking-Klassen und Normalen-Klassen mehr voneinander abgebildet werden, um das UML übersichtlicher zu gestalten.
2. **KISS (Keep It Simple, Stupid)**: Das Programm soll so einfach wie möglich gehalten werden.
   * **Beobachtung**: Die Start-Klasse ist einfach gehalten, aber könnte vereinfacht werden. Im Ganzen sind auch alle Methoden und Klassen logisch benannt.
   * **Verbesserung**: In der Start-Klasse hätte man einen Konstruktor implementieren können, der den User abfragt und die ausgewählte Klasse initialisiert. Ebenfalls statt eines Switch und Case hätte man auch ein if-Statement brauchen können, um den Code zu vereinfachen.
3. **YAGNI (You Aren't Gonna Need It)**: Füge keine Funktionalität hinzu, bevor sie notwendig ist.
   * **Beobachtung**: Der Code beschränkt sich auf die benötigte Funktionalität ausser GenerateNeededObject() in der Klasse Start.
   * **Verbesserung:** Die Logik der Methode GenerateNeededObject() in einen Konstruktor implementieren. Dazu eine Methode, die die Benutzereingabe behandelt, implementieren.
4. **DRY (Don't Repeat Yourself)**: Vermeide die Duplizierung von Code.
   * **Beobachtung**: Die Methoden **CollectData()** und **Calculate()** könnten beim Hinzufügen weiterer Formen zu Code-Duplikation führen.
   * **Verbesserung**: Eine Klasse DataHandler erstellen, die sich um das Sammeln von Daten und der Berechnung der jeweiligen Formen kümmert.
5. **Boochs Prinzipien (Coupling, Cohesion, Sufficiency, Completeness, Primitiveness)**: Strebe nach geringer Kopplung, hoher Kohäsion und stelle sicher, dass Klassen ausreichend, vollständig und primitiv sind.
   * **Beobachtung**: Die Klassen sind einzeln verantwortlich und sind primitiv. Die Kopplung könnte vereinzelt von der Klasse Controller jedoch verbessert werden.
   * **Verbesserung**: Entkopple die Controller-Klasse von der Hauptlogik der Form-Klassen von der Methode Calculate().
6. **SOLID-Prinzipien**:
   * **Single Responsibility**: Jede Klasse soll nur eine Verantwortung haben

**Beobachtung:** Die Start-Klasse ist für eine Benutzereingabe für die Objektgenerierung verantwortlich, somit hat die Klasse zwei Verantwortungen.

**Verbesserung:** Die Objektinitialisierung von der Benutzereingabe in der Klasse Start voneinander trennen. Eine weitere Klasse erstellen und die Benutzereingabe dort implementieren.

* + **Open-Closed**: Klassen sollen offen für Erweiterungen und verschlossen für Änderungen sein.

**Beobachtung**: der Code ist gut erweiterbar, dennoch muss man beachten, dass bei einer Klassenerweiterung von einer Form die Klasse Start angepasst werden.

* + **Liskov Substitution**: Abgeleitete Klassen sollen sich wie Elternklassen verhalten. Kinderklasse **is-a** Elternklasse! Keine Überraschungen.

**Beobachtung**: Der Code hält die Liskov Substitution ein. Das Interface Ishape ist richtig implementiert.

* + **Interface Segregation:** Möglichst wenig Interface (public) Methoden. Sollte übersichtlich bleiben. Hilft, die Klasse einfacher und damit weniger fehleranfällig zu halten. Hilft Anwendern der Klasse

**Beobachtung:** Das Interface hat wenige Methoden, ein benötigtes Property und ist kurzgehalten, somit hält es die Interface-Segregation ein.

* + **Dependency Inversion**: Klassen hoher Ebenen sollten nicht von Klassen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen. Einfache (aber sehr allgemeine) Regel: Kein „new“ für andere Klassen verwenden.
  + **Beobachtung**: Das **IShape** Interface erfüllt das Inversion-Prinzip, da es einfach gehalten wurde und kein «new» für die Generierung eines Objekts im zentralen genutzt wird und durch die Injektion wird die Kontrolle von einer Klasse an einer anderen Klasse übergeben.

1. **Gesetz von Demeter**: Ein Modul sollte die inneren Details der Objekte, mit denen es umgeht, nicht kennen.
   * **Beobachtung**: Die Controller-Klasse kennt das Interface, das Interface kennt aber die Controller-Klasse nicht. Man kann beliebig an den Form-Klassen etwas ändern. Controller muss dann nicht zwingend angepasst werden, ausser man möchte die Klasse vom Controller ansprechen.
   * **Verbesserung**: Der Controller führt jede Methode in der jeweiligen Form aus, dies könnte zu Komplikationen führen. Somit sollte man dies voneinander trennen, dass jede Klasse selbständig ist.

### *Verbesserung des Codes 4.2*

* **Boochs Prinzipien: Die Controllerklasse wirft nun nur die Methode CollectData in der vorher ausgewählten Formklasse auf. Somit ist die Controllerklasse vollständig von der Hauptlogik der Form-Klassen getrennt.**

**Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung**

**Circle-Klasse**

**Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung**

**Square-Klasse**

**Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung**

**Alle Tests geben nach Anpassung positives Ergebnis.**

**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung**

* **YAGNI: Da es ein Muss ist, dass ein Form-Objekt generiert wird. Habe ich nun die Initialisierung der Objekte in dem Start-Constructor implementiert. Dazu gibt es für die Benutzereingabe eine Methode, die die Eingaben behandelt.**

**Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung**

Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung

**Alle Tests geben nach Anpassung positives Ergebnis.**

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung

## Teilaufgabe 5

### *Unittest 5.1*

Die Klassen Square und Circle haben beide unterschiedliche Logiken, um die Attribute (Flächen, Umfang, etc.) zu berechnen. Um dies auf die Korrektheit zu prüfen, habe ich für die Methode Calculate() jeweils für die Klassen Square und Circle Unittests geschrieben. Die Unittests sind in der projektmapp im TestProject unter UnitTests.cs zu finden.

Ein Bild, das Text, Elektronik, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

## Teilaufgabe 6

### *Erklärung Selbstgemachter logischer Fehler 6.1*

Mein Programm gibt mit dem Nutzerinput 50 bei jeder Form unlogische Werte zurück, dazu wird der Durchmesser, Radius und Oberfläche von der Form Kreis nicht richtig berechnet.



Ein Bild, das Text, Schrift, Screenshot, Typografie enthält.

Automatisch generierte Beschreibung

Ein Bild, das Screenshot, Text, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

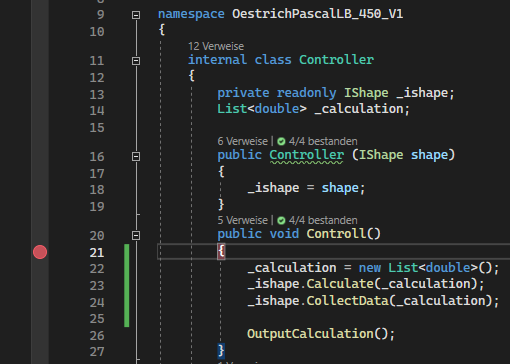
Automatisch generierte Beschreibung

Dies sind logische Fehler. Um logische Fehler zu beheben, kann man in Visual Studio den Debugging-Modus aktivieren und einzelne Schritte, die das Programm unternimmt, nachverfolgen. Den Debugging-Modus aktiviert man mit der Taste F11.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

So sieht es aus, wenn man einzeln Abläufe vom Programm nachvollziehen möchte. Aber da ich mir sicher bin, dass der Fehler in der Klasse Controller und bei in der Methode Caluclate() in der Klasse Circle sein muss, setze ich Breakpoints.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Mit Breakpoints kann man nun das Programm starten und es wird bei den vorher gesetzten Breakpoints einen Break setzen (Das Programm bleibt genau dort stehen). Nun kann man wieder mit F11 kleineren Schritten vom Programm nachvollziehen oder direkt zum nächsten Breakpoint springen.

Dazu sieht man im unteren Fenster (Autos) von Visual Studio alle Variablen, Collection und Objekte, die initialisiert wurden und welchen Wert sie bekommen haben. Ebenfalls sieht man noch die Werte und Variabel etc. die beim Start initialisiert wurden. Die Collection sieht man, welcher Index besetzt wurde und mit welchem Wert.

Ein Bild, das Text, Software, Multimedia-Software, Reihe enthält.

Automatisch generierte Beschreibung

Im Fenster Lokal sieht man nur die Variablen, Collections, Objekte das Lokal gerade gedebugged werden. Normalerweise sieht man dann die Variablen etc. die in der Methode.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

### *Unterschied zwischen Autos und Lokal 6.2*

* Autos: Im Fenster sieht man alle Variablen, Objekte und Collections in der ganzen Klasse, die gedebugged wird. Und Variablen etc. die in der aktuellen Codeabschnitt relevant sind.
* Lokal: Im Fenster Lokal sieht man nur die Variablen, Objekte und Collections, die gerade lokal gedebugged werden. (Zum Beispiel nur Werte in einer Methode, die gerade gedebugged wird).

Lokal ist sehr praktisch. Beim richtigen Verwenden macht es das Debuggen deutlich einfacher und übersichtlicher.

### *Watches 6.3*

Watches helfen das Debuggen deutlich übersichtlicher zu machen und ermöglicht ein einfacheres Testen.

Im Watch-Fenster kann man die Variablen, Objekte und Collections eingeben, die man anschauen beziehungsweise testen möchte.

Ein Bild, das Text, Software, Multimedia-Software, Screenshot enthält.

Automatisch generierte Beschreibung

Dazu kann man auch eigene Rechnungen eingeben und überprüfen, ob das Resultat mit der Logikrechnung vom Programm übereinstimmt.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

### *Debuggen 6.4*

Nachdem ich die Breakpoints gelegt habe, musste ich eine Nutzereingabe tätigen, hier ist noch alles normal.

Beim Breakpoint ging ich einzelne schritte durch und sah in der Methode Controll() das \_ishape.Calculate() dort nicht aufgerufen werden darf. Da noch vorher noch bei der Methode CollectData() ein Nutzerinput stattfinden muss, um damit die Seitenlänge oder den Umfang anzugeben, damit später gerechnet wird. Der Nutzerinput hat nicht stattgefunden, somit ist \_circumference = 0. Ich habe direkt den Fehler behoben.

Ein Bild, das Text, Screenshot, Schrift, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Behoben:

Ein Bild, das Text, Multimedia-Software, Schrift, Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Nun gehe ich weiter und erkenne, dass bei den Berechnungen die Flaschen Operationen stehen.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Software, Multimedia-Software, Screenshot enthält.

Automatisch generierte Beschreibung

Ein Bild, das Screenshot, Text, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Die Werte die ich in der Watch festgelegt habe, stimmen nicht mit den Werten im Programm überein.

Behoben:

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Software, Multimedia-Software, Screenshot enthält.

Automatisch generierte Beschreibung

Ein Bild, das Screenshot, Text, Multimedia-Software, Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Beim Starten des Unittests erkennt man, dass die logischen Fehler behoben wurden. Dazu beim Durchführen des Programms mit der Nutzereingabe 50 ist alles korrekt.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Somit war mein Debugging erfolgreich, da mein Programm nun funktioniert.

## Teilaufgabe 7

Im Test Driven Developement geht es darum, zuerst im Hauptsystem die Grundstruktur, also das, was man im UML sieht, zu implementieren. Und nachdem die Unittests oder Mocking-Klassen zu realisieren und auf der zuvor gecodeten Grundstruktur mithilfe der Test-Klassen und Methoden das eigentliche System zu programmieren. Ich bin so in meinem Programm für die Klassen Formen so vorgegangen. Zuerst habe ich die Grundstruktur in meinem Programm erstellt, danach die Tests (Mocking-Klassen, Unittest) und zum Schluss die Logik im Hauptsystem mit Hilfe der Test-Klassen und Methoden programmiert.

### *Klasse: Square 7.1*

Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**MockSpy-Klasse: Square**

**Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung**

**MockStub-Klasse: Square**

**Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**Unittest TestCalculationSquare(): Square**

**Ein Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung**

### *Klasse: Circle 7.2*

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Software, Betriebssystem enthält.

Automatisch generierte BeschreibungMockSpy-Klasse: Circle**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**MockStub-Klasse: Circle**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung**

**TestCalculationSquare(): Circle**

**Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung**

## Teilaufgabe 8

### *Implementierung Fehlerumgang 8.1*

Zum Schluss habe ich noch das Programm so modifiziert, dass es mit Fehlereingaben umgehen kann und den Nutzer auf die Fehlereingabe hinweist. Dazu habe ich in der Klasse Square eine weitere Methode implementiert, die zwei Eingaben auf logische Fehler überprüft. Diese Methode ist im aktualisierten UML unten ersichtlich.

Ein Bild, das Text, Diagramm, parallel, Zahl enthält.

Automatisch generierte Beschreibung

### *Testprotokoll: 8.2*

|  |  |  |  |
| --- | --- | --- | --- |
| **TC-№** | **Datum** | **Resultat** | **Tester** |
| 1.1 | 12.01.2024 | OK | Pascal Oestrich |
| 2.1 | 12.01.2024 | OK | Pascal Oestrich |
| 3.1 | 12.01.2024 | OK | Pascal Oestrich |
| 4.1 | 12.01.2024 | OK | Pascal Oestrich |
| 5.1 | 12.01.2024 | OK | Pascal Oestrich |
| 5.2 | 12.01.2024 | OK | Pascal Oestrich |
| 5.3 | 12.01.2024 | OK | Pascal Oestrich |
| 5.4 | 12.01.2024 | OK | Pascal Oestrich |
| 6.1 | 12.01.2024 | OK | Pascal Oestrich |
| 6.2 | 12.01.2024 | OK | Pascal Oestrich |
| 7.1 | 12.01.2024 | OK | Pascal Oestrich |
| 8.1 | 12.01.2024 | OK | Pascal Oestrich |

### *Testfazit 8.3*

Fazit: Alle Tests haben alle ein positives Ergebnis, Entwicklung beendet, Programm kann für den für den Kunden (LB) freigegeben werden.